

CBP-NG: the Next-Generation Championship in Branch Prediction

Pierre Michaud

PC chair

June 28, 2026

outline

- overview of HARCOT
- VFS scoring metric
- example predictors
- submissions and selection process

HARCOM

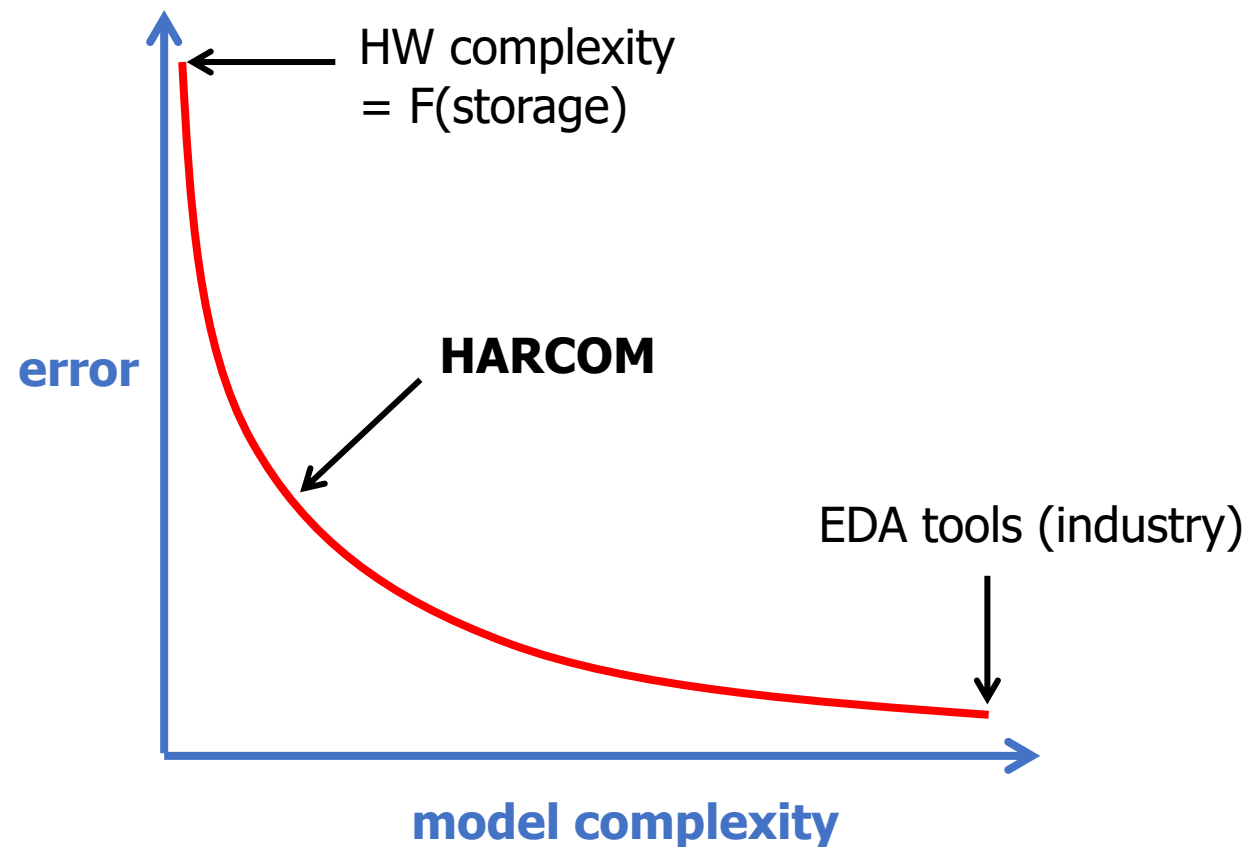
hardware **c**omplexity **m**odel for microarchitecture exploration

HARCOM, briefly

- domain-specific language embedded into C++
 - single header file harcom.hpp
 - manual < 50 pages (but you need to know some C++)
- level of abstraction close to RTL
 - but not a hardware description language (no netlist generated)
- can be used inside a microarchitecture simulator (gem5, ChampSim, ...)
 - component of interest modeled with HARCOM; rest of simulator unmodified
- functional simulation produces estimate of hardware complexity
 - delays, # transistors, energy

medium-complexity model

- many guesses
 - ~ 5nm FinFET
 - I did not have access to a commercial PDK
- many approximations
- many arbitrary choices
- **I do not know the error of HARCOT**



data types

- **val<N>**
 - N-bit transient integer value; **N ≤ 64**
 - cannot be modified
- **reg<N>**
 - persistent val<N> that can be modified once per clock cycle
- **arr<T,M>**
 - array of M values of type **T** = val<N> or **T** = reg<N>
- **hard<N>**
 - fixed integer, known at compile time

operations on vals/regs produce vals

- most operators of the C language

```
reg<1> x = 1;
```

```
val<1> y = x ^ x; // xor
```

```
reg<2> z = x + x; // add
```

...

- multiple member functions

```
reg<7> x7 = 42;
```

```
val<1> x1 = x7.make_array(val<1>{}).fold_xor(); // split x7 into arr<val<1>,7> and xor the 7 bits
```

```
arr<val<1>,7> a7 = x1.replicate(hard<7>{}); // replicate x1 seven times
```

```
arr<val<1>,128> a128 = x7.decode(); // decode x7 (one hot)
```

...

conditional execution

- `execute_if(cond, action)`
 - `cond = val<1>`
 - `action = C++ lambda`
- example

```
reg<1> x = 3;  
val<1> cond = (x == hard<3>{});  
execute_if(cond, [&]() { x = 2; });
```
- no energy consumption if the condition is false

random access memory

- `ram<T,M>`
 - M entries
 - data port of type `T = val<N>` or `T = arr<val<N>,M>`
- example:

```
ram<val<32>, 16> bank; // RAM with 16 entries, data port of type val<32>
val<4> addr = 13;
val<32> data = 0xffffffff;
bank.write(addr,data); // write the RAM
data = bank.read(addr); // read the RAM
```
- can be accessed (read or write) once per clock cycle

HARCOM panel

- provides access to various stats

- total energy
- total # transistors
- total storage
- ...

- example:

```
val<64> x = 0;
```

```
x+1;
```

```
panel. print(); // prints all stats
```

HARCOM superuser

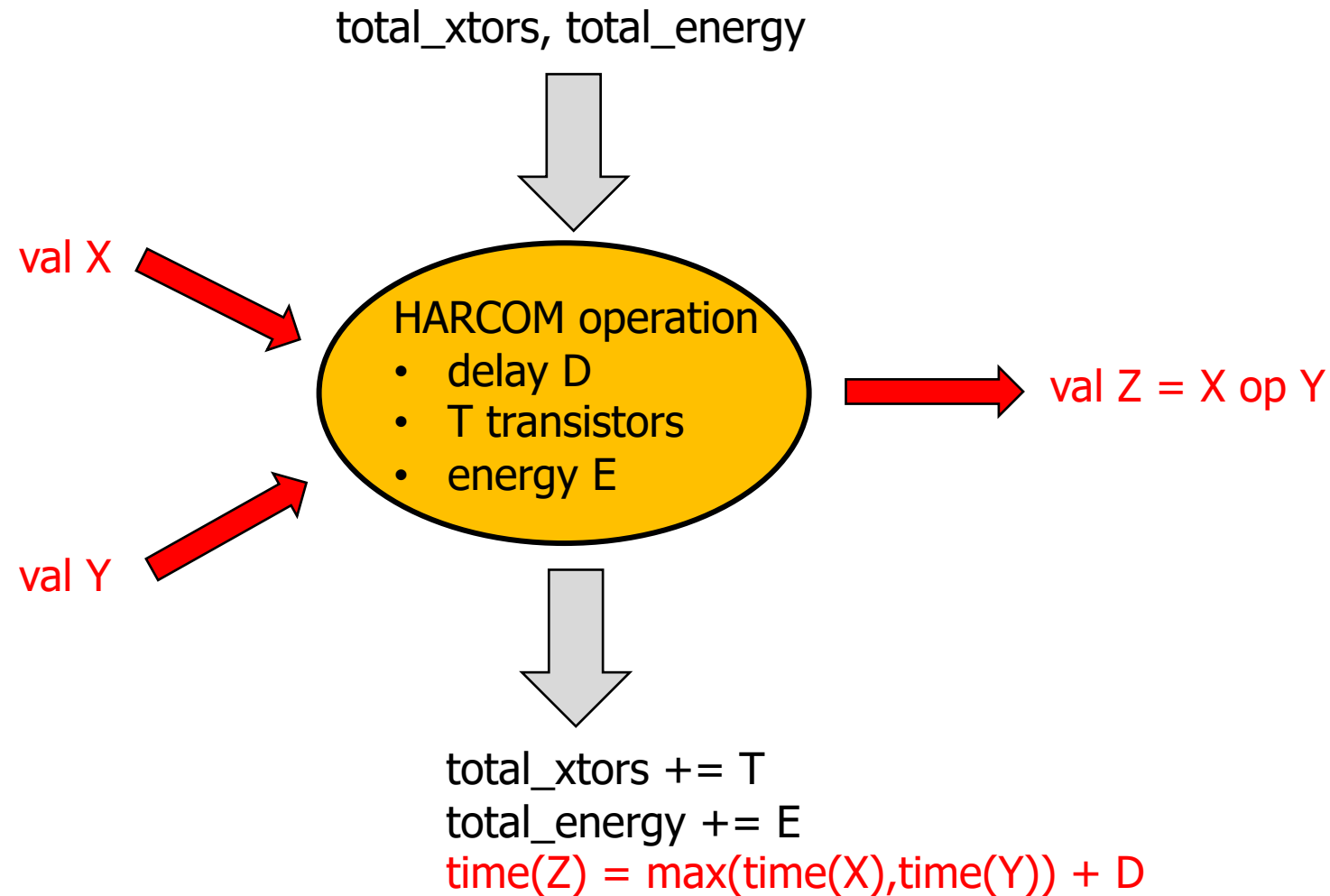
- class `harcom_superuser`
 - can convert HARCOM vals into C++ integers
 - can inspect and set the timing of vals
 - can increment the clock cycle (call to `panel.next_cycle()`)
- the CBP-NG simulator is an object of type `harcom_superuser`
 - contestants are not allowed to modify it

hardware complexity model

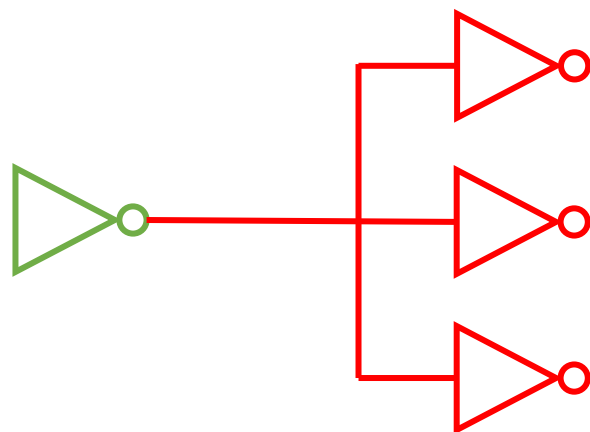
- similar to CACTI and McPAT
 - transistor and wire behavior modeled with resistances and capacitances
- every statement in the HARCOT language has a hardware cost
 - HW cost = delay, # transistors, energy
- the HW cost can be null for certain operations

```
val<8> x = 0b10010110;  
val<8> y = x << 1; // left shift has null HW cost
```

hardware cost updated on each operation



gate delay increases linearly with fanout



$$\text{gate delay} = \left(c_p + \frac{c_l}{s} \right) \times \tau$$

parasitic capacitance (dimensionless)

gate size (dimensionless)

load capacitance (dimensionless)

transistor intrinsic delay

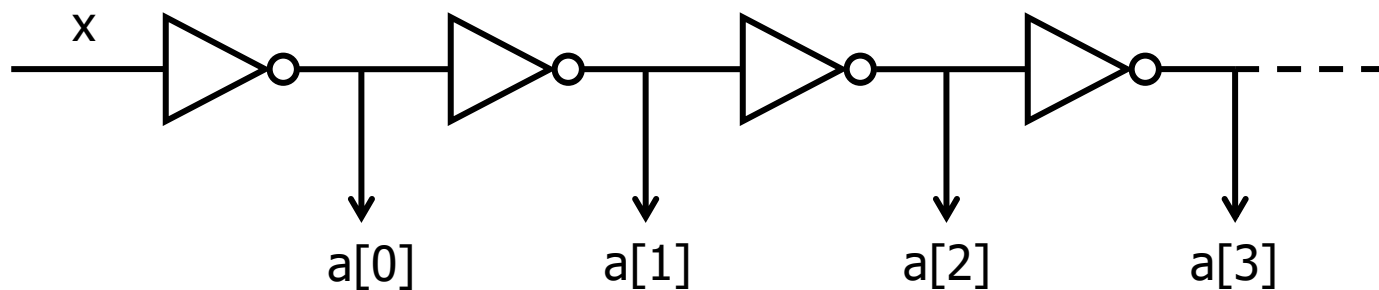
the HW cost of reading vals

```
val<1> x = 1;  
arr<val<1>,4> a = {x,x,x,x}; // read x 4 times  
a.print("a");  
panel.transistors().print("xtors: ");
```



```
a0: 1 (t=4 ps, loc=0)  
a1: 1 (t=8 ps, loc=0)  
a2: 1 (t=12 ps, loc=0)  
a3: 1 (t=16 ps, loc=0)  
xtors: 8
```

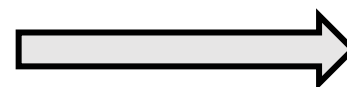
each read → HW cost of a FO2 inverter



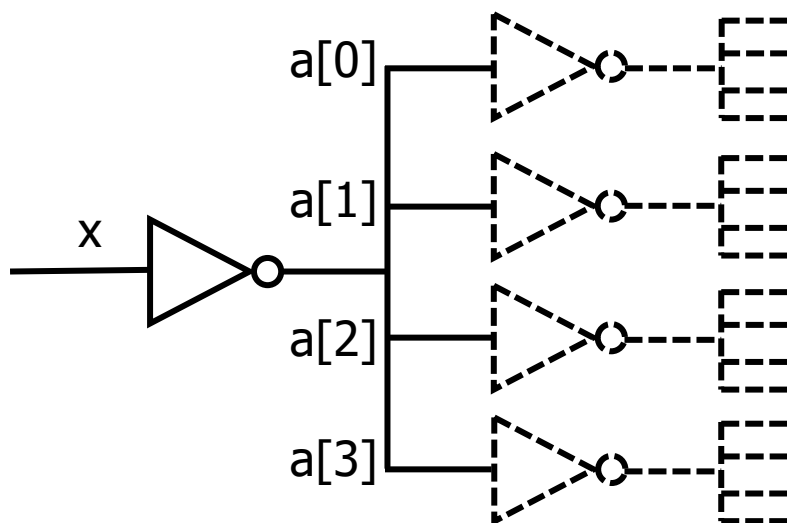
**delay increases linearly
with number of reads**

declaring the fanout

```
val<1> x = 1;
x.fanout(hard<4>{}); // declare the fanout of x
arr<val<1>,4> a = {x,x,x,x};
a.print("a");
panel.transistors().print("xtors: ");
```



```
a0: 1 (t=6 ps, loc=0)
a1: 1 (t=6 ps, loc=0)
a2: 1 (t=6 ps, loc=0)
a3: 1 (t=6 ps, loc=0)
xtors: 2
```



tree of FO4 inverters

➔ **delay increases logarithmically with fanout**

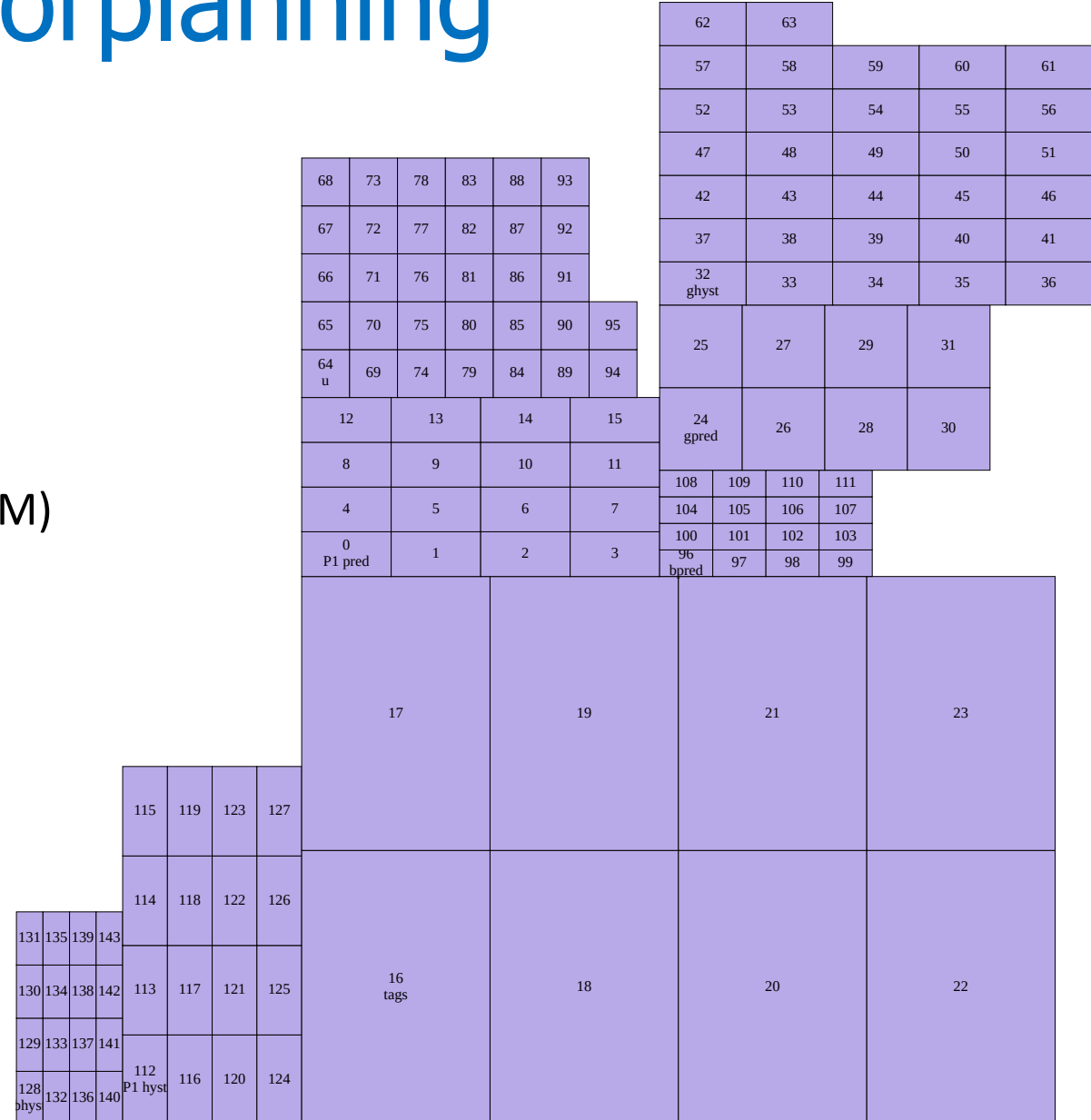
static RAM

- ram<T,M> modeled as high density (6T) **single R/W port** SRAM
- large SRAM divided into multiple subarrays connected via H-tree networks
 - somewhat similar to CACTI
- explore a few configurations (subarray rows/columns)
 - select the one with minimum energy times latency cubed ($E \times D^3$)
- ideal pipelining: one access per clock cycle
 - latches/flip-flops inside SRAM not modeled
- rectangle shape



floorplanning

- primitive floorplanning algorithm
- only SRAMs
 - area of combinational logic (outside SRAM) and registers not modeled



location of a val

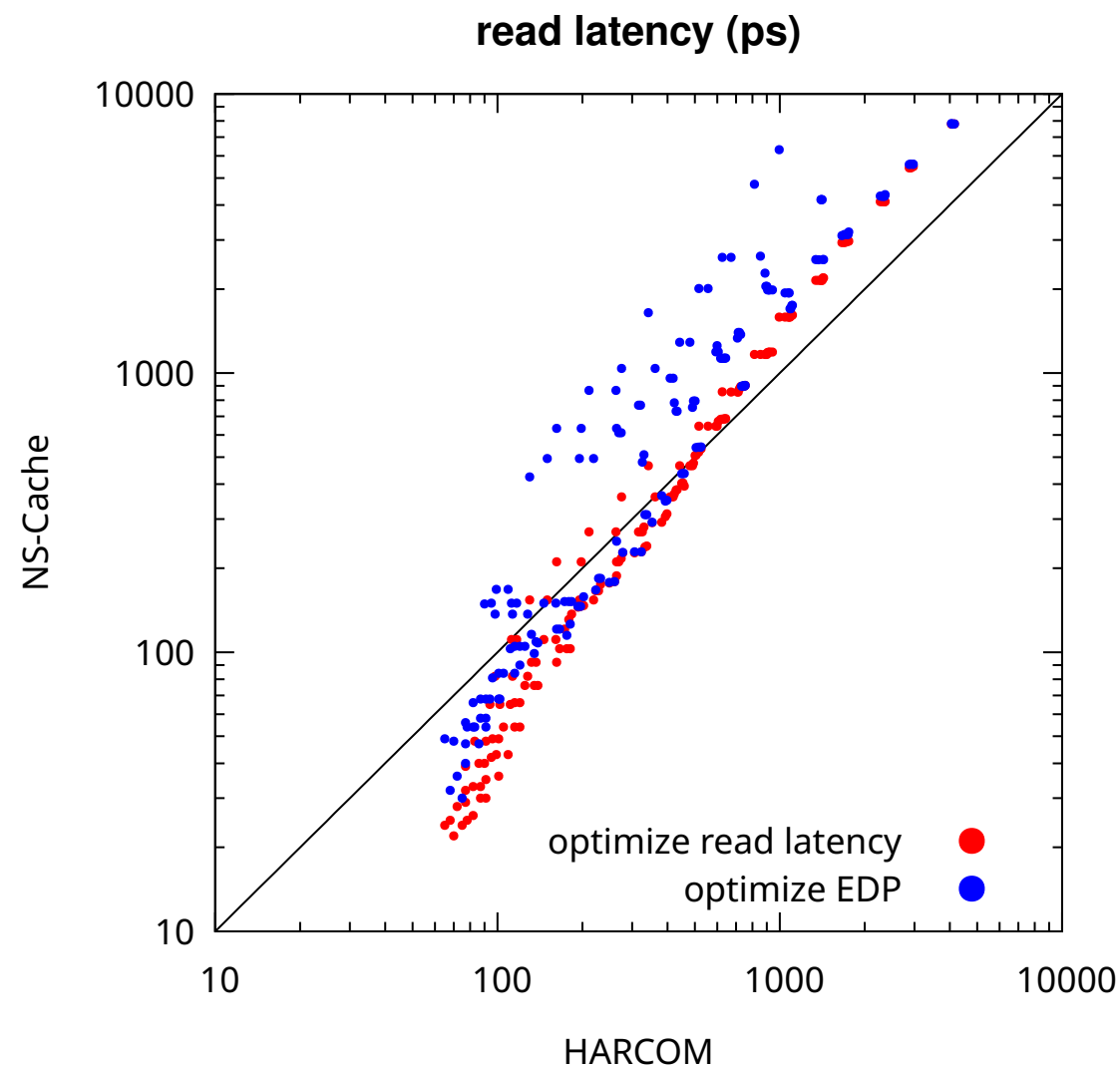
- each val has a location, which is one of the RAMs
- a reg is located at the previous RAM in declaration order
- **locus** of an operation:
 - RAM operation (R/W) → locus is this RAM
 - non-RAM operation → locus is location of latest arriving input
- the location of a val produced by an operation is this operation's locus

HW cost of wires

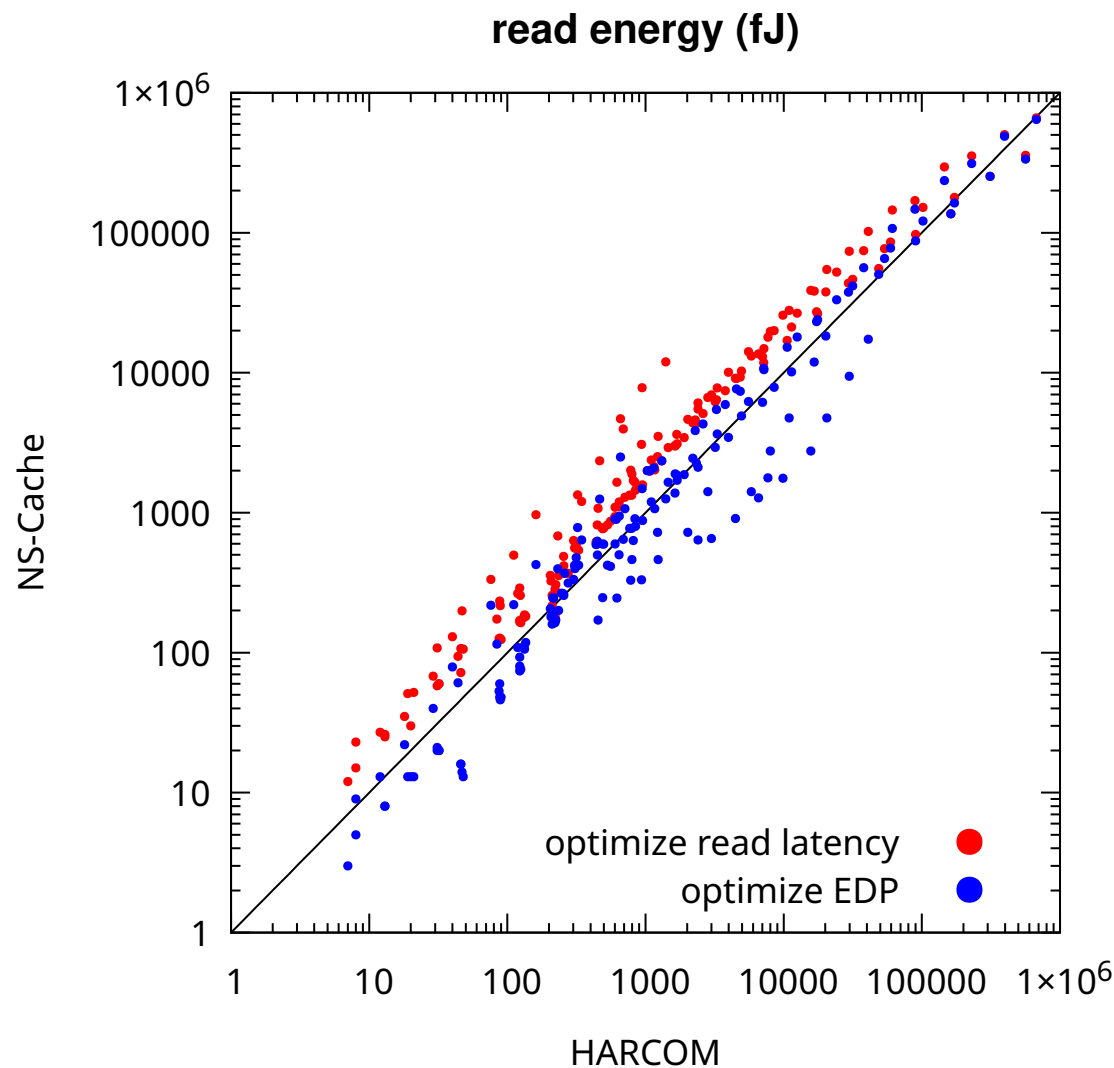
- distant input = input whose location is different from the locus of the operation
- assume distinct wire for each bit of distant inputs
 - HARCOM models only one type of wire (semi-global)
- HW cost of wire depends on Manhattan distance
- long wire segmented with repeaters

SRAM read latency vs. NS-Cache

- NS-cache is a cache model from Georgia Tech
 - *Optimization and Benchmarking of Monolithically Stackable Gain Cell Memory for Last-Level Cache*, Waqar, Kwak, Lee, Phadke, Shon, Gholamrezaei, Skadron & Yu, IEEE TC, vol. 75, no. 3, March 2026
 - **derived from CACTI**
- compare with NS-Cache's 5nm node
- vary SRAM entries and data size
 - 2^E entries, E in $\{6, \dots, 22\}$
 - 2^D bits per entry, D in $\{0, \dots, 9\}$
 - ➔ 170 configurations



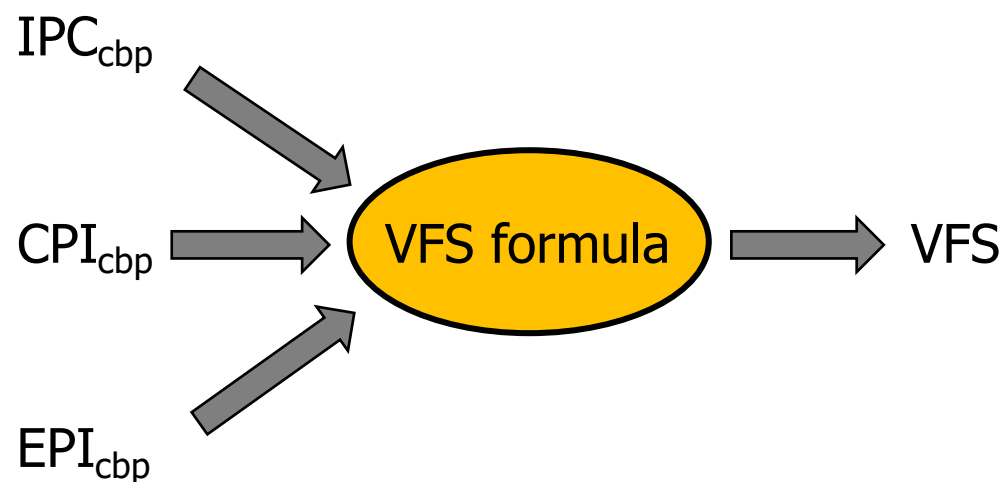
SRAM read energy vs. NS-Cache



the VFS scoring metric

the VFS, briefly

- **v**oltage/**f**requency-scaled **s**peedup
- mathematical formula corresponding to an ultra-simplified performance model
- 3 inputs: IPC_{cbp} , CPI_{cbp} , EPI_{cbp}
 - **cbp** = conditional branch predictor
- output = VFS number
 - higher is better



the three figures of merit (FoM)

- IPC_{cbp} (“predictor throughput”)
 - number of instructions predicted per cycle
 - harmonic mean over all traces
- CPI_{cbp} (“predictor performance”)
 - L2 mispredictions per correct-path instruction times misprediction penalty
 - arithmetic mean over all traces
- EPI_{cbp} (“predictor energy”)
 - predictor energy per correct-path instruction
 - arithmetic mean over all traces

assumption: the CBP is the IPC bottleneck

- except for the CBP, the core frontend and backend are ideal

$$\text{IPC} = \frac{1}{\frac{1}{\text{IPC}_{\text{cbp}}} + \text{CPI}_{\text{cbp}}}$$

- **the core microarchitecture depends on the CBP**
 - in reality, it is the other way around

reference predictor

- $IPC^*_{cbp}, CPI^*_{cbp}, EPI^*_{cbp}$  $IPC^* = \frac{1}{\frac{1}{IPC^*_{cbp}} + CPI^*_{cbp}}$

- defined by the organizers

- **predictor we want**

- not a predictor we already have

assumption: fixed power budget

- adjust voltage & frequency → core power equals power P^* of reference core
 - consider switching power only
 - assume that IPC does not vary with clock frequency
 - model clock frequency vs. voltage

frequency scaling factor

$$VFS = \frac{IPC}{IPC^*} \times \left(1 - \frac{2}{1 + \sqrt{1 + \beta \times \frac{P^*}{P}}} \right) \times \alpha$$

technology parameter

$\beta = \frac{4\alpha}{(\alpha - 1)^2}$

core power before V/F scaling

core energy model

$$EPI = \left[EPI_{cbp} + E_C^* \times \left(\frac{IPC}{IPC^*} \right)^\gamma \right] \times \left(1 + \frac{IPC_{cbp} \times CPI_{cbp}}{2} \right)$$

$\gamma = \frac{2}{\alpha - 1}$

wrong-path instructions per correct-path instruction

core energy per correct-path instruction (before V/F scaling)

non-CBP energy of a correct-path instruction in the reference core

assume that the energy of a wrong-path instruction is half that of a correct-path one

the VFS is an objective function

- we define an ambitious reference predictor
 - $IPC_{cbp}^* = 8$ instructions per cycle (multiple not-taken branches)
 - $CPI_{cbp}^* = 0.0315$ cycles per instruction → Seznec's 192KB TAGE-SC with 1-cycle latency
 - $EPI_{cbp}^* = 1$ pJ per instruction
- $VFS^* = 1$
 - $VFS < 1$ for all the predictors submitted to CBP-NG
- IPC_{cbp} must be close to IPC_{cbp}^* for a predictor to be competitive

example predictors



 main

cbp-ng / predictors



Go to file

...

 Pierre Michaud and alindsay-ampere

Change the polarity of the hysteresis bit in gshareN_ahead.

0f3433c · 2 months ago

History

Name	Last commit message	Last commit date
..		
tutorial	Tutorial: Replace std::format usage with ostreamstream	4 months ago
always_taken.hpp	New interface for update_cycle():	4 months ago
bimodal.hpp	Change the definition and behavior of execute_if.	3 months ago
bimodalN.hpp	Change the definition and behavior of execute_if.	3 months ago
common.hpp	Change the definition and behavior of execute_if.	3 months ago
gshare.hpp	Change the definition and behavior of execute_if.	3 months ago
gshareN.hpp	Change the definition and behavior of execute_if.	3 months ago
gshareN_ahead.hpp	Change the polarity of the hysteresis bit in gshareN_ahead.	2 months ago
hashed_perceptron.hpp	Add a hashed perceptron branch predictor	3 months ago
never_taken.hpp	New interface for update_cycle():	4 months ago
perceptron.hpp	Change the definition and behavior of execute_if.	3 months ago
tage.hpp	Change the definition and behavior of execute_if.	3 months ago

Pierre, Aaron, lpha

TAGE example

- basic TAGE; not optimized
- 35 KB; 8 tagged tables; global history of 100 branches
- latency = 2 cycles; driven by single-cycle gshare
 - throughput loss when gshare and TAGE disagree
- index = line address; predict all 16 instructions in 64B line
 - predictions before line entry point or after taken branch are not used
 - interleaved bimodal table → 16 bimodal predictions
 - tags extended with 4-bit line offset
- $EPI_{cbp}^* = 1$ pJ represents 80% of this TAGE's EPI_{cbp}

gshareN_ahead

- 128 KB; global history of 16 branches
- **ahead pipelined** (self driven)
 - A. Seznec et al., ASPLOS 1996; D. A. Jiménez, HPCA 2003
- index = block address
 - *Branch History Table Indexing to Prevent Pipeline Bubbles in Wide-Issue Superscalar Processors*, T.-Y. Yeh and Y. N. Patt, MICRO 1993
- branches inside same block are distinguished by their **rank**
 - up to 7 branches per block; 8 lanes; rotate lanes for uniform use
 - *Exploring Instruction-Fetch Bandwidth Requirement in Wide-Issue Superscalar Processors*, P. Michaud, A. Seznec & S. Jourdan, PACT 1999
 - see André Seznec's [cookbook](#) for rank-based TAGE
- rank-based prediction: more research is needed
 - rank-based BTB, checkpointing, rank computation, update,...

submissions and selection

program committee

- Pierre Michaud (Inria), Chair
- Alaa Alameldeen (Simon Fraser University)
- Daniel Jiménez (Texas A&M University)
- Eric Rotenberg (North Carolina State University)
- Gilles Pokam (AMD)
- Houdhaifa Bouzguarrou (ARM)
- Michael Chin (Ampere)
- Rami Sheikh (ARM)
- Syed Fakhri (Ampere)
- Thibaut Lanois (ARM)

submissions, reviews & selection

- 9 submissions
- 3 reviews per paper; 3 papers per reviewer
- reviewers had only one week to do the reviews
 - no access to submitted codes
- I selected submissions with VFS above 0.9 or more accepts than rejects
- → **7 submissions** selected for the workshop

CBP-NG program

- Session 1 (➔ 3:30 pm)
 - **An Energy-Efficient Ahead-Pipelined TAGE Branch Predictor for CBP-NG** (Nhat Dang & Eric Rotenberg)
 - **Enhance Ahead-Pipelined N-branch GShare with Tagged Tables** (Jun Fan)
 - **Distilled Branch Predictors** (Simha Sethumadhavan)
 - **RABT - Run-Ahead Block TAGE** (Prakhar Gupta et al.)
- Session 2 (4:00 pm ➔)
 - **MORSL: Minimal-Overhead Rank-Based Predictor with Summation-Free Correction and Lazy Access** (Toru Koizumi et al.)
 - **Coding Agents as Design Searchers: An Autonomous TAGE Tuning Campaign for CBP-NG** (Matt Pallan)
 - **Offset-Free TAGE-SC: Conditional-Branch-Level Prediction for Wide Fetch Frontends** (Yashwant Kumar Balivada & Sairam Viswanathan Susarla)